Consensus and Reliable Broadcast

The Reliable Broadcast Problem

- **Validity**: If the sender is correct and broadcasts a message \( m \), then all correct processes eventually deliver \( m \).
- **Agreement**: If a correct process delivers a message \( m \), then all correct processes eventually deliver \( m \).
- **Integrity**: Every correct process delivers at most one message, and if it delivers \( m \), then some process must have broadcast \( m \).

The Terminating Reliable Broadcast Problem

- **Termination**: Every correct process eventually delivers some message.
- **Validity**: If a correct process broadcasts a message \( m \), then all correct processes eventually deliver \( m \).
- **Agreement**: If a correct process delivers a message \( m \), then all correct processes eventually deliver \( m \).
- **Integrity**: Every correct process delivers at most one message, and, if it delivers \( m \) \( \neq \text{SF} \), then some process must have broadcast \( m \).

Broadcast

**BC**: If a process sends a message \( m \), then every process eventually delivers \( m \).

Can we implement this specification if processes can fail?
### The Consensus Problem

- **Termination**: Every correct process eventually decides some value.
- **Validity**: If all processes that propose a value propose \( v \), then all correct processes eventually decide \( v \).
- **Agreement**: If a correct process decides \( v \), then all correct processes eventually decide \( v \).
- **Integrity**: Every correct process decides at most one value, and if it decides \( v \neq \text{NU} \), then some process must have proposed \( v \).

### Properties of `send(m)` and `receive(m)`

For benign failures:

- **Validity**: If \( p \) sends \( m \) to \( q \), and both \( p \) and \( q \) and the link between them are correct, then \( q \) eventually receives \( m \).
- **Uniform Integrity**: For any message \( m \), \( q \) receives \( m \) at most once from \( p \), and only if \( p \) sent \( m \) to \( q \).

For arbitrary failures:

- **Integrity**: For any message \( m \), if \( p \) and \( q \) are correct then \( q \) receives \( m \) at most once from \( p \), and only if \( p \) sent \( m \) to \( q \).

### Questions, Questions…

- Are these problems solvable at all?
- Can they be solved independent of the failure model?
- Does solvability depend on the ratio between faulty and correct processes?
- Does solvability depend on assumptions about the reliability of the network?
- Are the problems solvable in both synchronous and asynchronous systems?
- If a solution exists, how expensive is it?

### Plan

**Synchronous Systems**

- Consensus for synchronous systems with crash failures
- Lower bound on the number of rounds
- Early stopping protocols for Reliable Broadcast
- Reliable Broadcast for arbitrary failures with message authentication
- Lower bound on the ratio of faulty processes for Consensus with arbitrary failures
- Reliable Broadcast for arbitrary failures

**Asynchronous Systems**

- Impossibility of Consensus for crash failures
**Model**

- Synchronous Message Passing
  - Execution is a sequence of rounds
  - In each round every process takes a step
    - sends messages to neighbors
    - receives messages sent in that round
    - changes its state
- Network is fully connected (an n-clique)
- No communication failures

**A simple algorithm for Consensus**

**Code for process $p_i$:**

```
Initially $V = \{v\}$

To execute $\text{propose}(v)$
1: send $(v)$ to all

$\text{decide}(v)$ occurs as follows:
2: for all $j$, $0 \leq j \leq n-1$, $j \neq i$ do
3: receive $S_j$ from $p_j$
4: $V = V \cup S_j$
5: $\text{decide } \min(V)$
```

**An execution**

- $v_1$, $v_2$, $v_3$, $v_4$
- Round 1
  - $p_1$, $p_2$, $p_3$, $p_4$
  - $p_1$, $p_2$, $p_3$, $p_4$
  - $v_1$, $v_2$, $v_3$, $v_4$
  - $v_1$, $v_2$, $v_3$, $v_4$

**Can $p_1$ decide**

- $v = v_1 = v_3 = v_4$?

**Idea**

- A process that receives a proposed message in round 1, relays it to others during the next round
- Suppose $p_i$ hasn’t heard from $p_j$ at the end of round 2. Can it decide?
In general…

- Suppose a correct process \( p^* \) has not received all proposals by the end of round \( i \). Can \( p^* \) decide?
- Another process may have received the missing proposal at the end of round \( i \) and be ready to relay it in round \( i + 1 \).

### Dangerous Chains

- How many rounds can a dangerous chain span?
  - \( f \) faulty processes
  - at most \( f + 1 \) nodes in the chain
  - spans at most \( f \) rounds

A dangerous chain: The last node in the chain is correct, all others are faulty

It is safe to decide after round \( f + 1 \)

The Algorithm

#### Code for process \( p_i \):

Initially \( V = \{v_i\} \)

To execute \( \text{propose}(v) \)

round \( k \), \( 1 \leq k \leq f+1 \)

1. send \( \{v \in V : p_i \text{ has not already sent } v\} \) to all
2. for all \( j, 0 \leq j \leq n-1, j \neq i \) do
3. receive \( S_j \) from \( p_j \)
4. \( V := V \cup S_j \)

\( \text{decide}(x) \) occurs as follows:

5. if \( k = f+1 \) then
6. \( \text{decide} \min(V) \)

#### The Algorithm

<table>
<thead>
<tr>
<th>Code for process ( p_i ):</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initially</strong> ( V = {v_i} )</td>
</tr>
<tr>
<td><strong>To execute</strong> ( \text{propose}(v) )</td>
</tr>
<tr>
<td><strong>round</strong> ( k ), ( 1 \leq k \leq f+1 )</td>
</tr>
<tr>
<td>1. send ( {v \in V : p_i \text{ has not already sent } v} ) to all</td>
</tr>
<tr>
<td>2. for all ( j, 0 \leq j \leq n-1, j \neq i ) do</td>
</tr>
<tr>
<td>3. receive ( S_j ) from ( p_j )</td>
</tr>
<tr>
<td>4. ( V := V \cup S_j )</td>
</tr>
<tr>
<td>( \text{decide}(x) ) occurs as follows:</td>
</tr>
<tr>
<td>5. if ( k = f+1 ) then</td>
</tr>
<tr>
<td>6. ( \text{decide} \min(V) )</td>
</tr>
</tbody>
</table>

**Termination and Integrity**

- **Integrity**
  - **Termination**
    - Every correct process reaches round \( f + 1 \)
    - \( \text{decides} \min(V) \), which is well defined
Validity

- Suppose every process proposes $v^*$
- Since only crash model, only $v^*$ can be sent
- By Uniform Integrity of send and receive, only $v^*$ can be received
- By protocol, $v = v^*$
- $\min(v) = v^*$
- $\text{decide}(v^*)$

A Lower Bound

Theorem

There is no algorithm that solves the consensus problem in less than $f + 1$ rounds in the presence of $f$ crash failures, if $n \geq f + 2$

- Prove special case $f = 1$ to study proof technique

Agreement

Lemma 1:

For any $r = 1, \ldots, f$, if a process $p$ receives a value $v$ in round $r$, then there exists a sequence of processes $p_{r_1}, p_{r_2}, \ldots, p_{r_{\ell}}$ such that $p_{r_{\ell}} \neq p$ (protocol $p^*$ is used) and in each round $i$, $\exists i \geq i \geq 1; p_{r_{i}}$, sends $v$ and $p_{r_{i}}$ receives it. Furthermore, all processes in the sequence are distinct.

Proof:

- Suppose a correct process $c$ in its view $V$ determines a value $v^*$
- $\text{decide}(v^*)$
- $\forall p \in V$, $\exists v^* \text{ s.t. } \text{decide}(v^*)$

Definition Let $\mathcal{E}$ be an execution and let $p_i$ be a process. The view of $p_i$ in $\mathcal{E}$, denoted by $\mathcal{V}_{p_i}$, is the subsequence of computation and message receive events that occur in $p_i$ together with the state of $p_i$ in the initial configuration of $\mathcal{E}$. Views
Similarity

**Definition** Let \( A \) and \( B \) be two executions of consensus and let \( p \) be a correct process in \( A \) and \( B \). Execution \( A \) is similar to execution \( B \) with respect to \( p \), denoted \( A \sim_p B \), if \( A(p) = B(p) \).

**Note** If \( A \sim_p B \), then \( p \) decides the same value in both executions.

**Lemma** If \( A \sim_p B \) and \( p \) is correct, then \( \text{dec}(A) = \text{dec}(B) \).

Single-Failure Case

**Theorem**
There is no algorithm that solves the consensus problem in less than 2 rounds in the presence of 1 crash failure, if \( n \geq 3 \).

The Idea

- Proceed by contradiction:
  - Consider an execution in which each process proposes 0. What is the decision value?
  - Consider another execution in which each process proposes 1. What is the decision value?
  - Show that there is a chain of similar executions that relate the two executions.
  - So what?

The Proof

**Definition** \( J \) is the admissible execution of the algorithm in which
- no failures occur
- processes \( p_0, \ldots, p_2 \) propose 1.
The Proof - 2

- We want to show that $\mathcal{G}': 0 \rightarrow p \rightarrow \mathcal{G}''$
- Starting from $\mathcal{G}$, we build a set of executions $\mathcal{G}'$, where $0 \rightarrow p \rightarrow n - 1$, as follows:

$\mathcal{G}'$ is obtained from $\mathcal{G}$ after removing the messages that $p_i$ sends to the $j$ highest numbered processors (excluding itself).

The executions

The Proof - 3: Indistinguishibility

The Terminating Reliable Broadcast Problem

- **Termination**: Every correct process eventually delivers some message
- **Validity**: If a correct process broadcasts a message $m$, then all correct processes eventually deliver $m$
- **Agreement**: If a correct process delivers a message $m$, then all correct processes eventually deliver $m$
- **Integrity**: Every correct process delivers at most one message, and, if it delivers $m \neq SF$, then some process must have broadcast $m$
Reliable Broadcast for Benign Failures

Terminates in $f + 1$ rounds
• even if there are no failures!

Can we do better?
• find a protocol whose time complexity is proportional to $t$—the number of failures that actually occurred—rather than to $f$—the max number of failures that may occur

What is the danger?

Valid Messages

A message is *valid* if it has the following form:
- in round 1:
  \[ < m, \text{sig}(s) > \text{ where } s \text{ is the sender} \]
- in round $r > 1$, if received by $p$ from $q$:
  \[ < \ldots (m, \text{sig}(p)), \text{sig}(p_1), \ldots, \text{sig}(p_r)) > \text{ where} \]
  - $p_1 = \text{sender}$, $p_r = q$
  - $p_1 \ldots p_r$ are distinct from each other and from $p$
  - message has not been tampered with

\[ < \ldots (m, \text{sig}(p_1)), \text{sig}(p_2), \ldots, \text{sig}(p_r)) > : \]
- in round $r$, $p_r$ said that in round $r - 1$, $p_{r-1}$ said that...
- ... in round 1, $p_1$ said $m$

AFMA: The Idea

• A correct process $p$ discard all non-valid messages it receives
• If a message is valid,
  - it “extracts” the value from the message
  - it relays the message, with its own signature appended
• At round $f + 1$:
  - if $p$ extracted exactly one message, delivers it
  - otherwise, delivers SF
AFMA: The Protocol

**Termination**
- In round \( f + 1 \), every correct process delivers either \( m \) or SF and then halts.

**Agreement**
- From Agreement and the observation that the sender, if correct, delivers its own message.

**Validity**
- From Agreement and the observation that the sender, if correct, delivers its own message.

---

**Proof**
Let \( r \) be the earliest round in which some correct process extracts \( m \). Let that process be \( p \).
- If \( p \) is the sender, then in round \( r \) sender sends a valid message to all. All correct processes extract message in round 1.
- Otherwise, \( p \) has received a valid message to round \( r \).

**Validity**
- From Agreement and the observation that the sender, if correct, delivers its own message.

**Termination**
- In round \( f + 1 \), every correct process delivers either \( m \) or SF and then halts.

---

**Observation**
- The sender, if correct, delivers its own message.

---

**Lemma**
If a correct process extracts \( m \), then every correct process eventually extracts \( m \).
TRB for Arbitrary Failures

AF: The Idea

- Identify the essential properties of message authentication that made AFMA work
- Implement these properties without using message authentication

TRB for Arbitrary Failures

AF: The Idea

• Introduce two primitives
  - \texttt{broadcast}(p,m,i) (executed by \( p \) in round \( i \))
  - \texttt{accept}(p,m,i) (executed by \( q \) in round \( j \geq i \))
• Give axiomatic definitions of broadcast and accept
• Derive an algorithm that solves TRB for AF using these primitives
• Show an implementation of these primitives that does not use message authentication

Properties of \texttt{broadcast} and \texttt{accept}

• **Correctness** If a correct process \( p \) executes \texttt{broadcast}(p,m,i) in round \( i \), then all correct processes will execute \texttt{accept}(p,m,i) in round \( i \)
• **Unforgeability** If a correct process \( q \) executes \texttt{accept}(p,m,i) in round \( j \geq i \), and \( p \) is correct, then \( p \) did in fact execute \texttt{broadcast}(p,m,i) in round \( i \)
• **Relay** If a correct process \( q \) executes \texttt{accept}(p,m,i) in round \( j \geq i \), then all correct processes will execute \texttt{accept}(p,m,i) by round \( j + 1 \)
AF: The Protocol - 1

Termination

- In round $f+1$, every correct process delivers either $m$ or $SF$ and then halts

Agreement -1

Proof

Let $r$ be the earliest round in which some correct process extracts $m$. Let that process be $p$. Then $p$ is executed in round $r$.

Claim 1: $p$ has accepted $m$. 

Suppose $p$ has not accepted $m$. Then $p$ has not previously extracted $m$. Hence $m$ is not delivered by any process in round $r$. By the claim, $m$ is not delivered in round $r$.

Claim 2: $p$ has not previously extracted $m$.

Suppose $p$ has previously extracted $m$. Then $m$ is delivered by some process $q$ in round $r$. But $q$ has not previously extracted $m$. Hence $m$ is not delivered by any process in round $r$.

Claim 3: $p$ has not previously extracted $m$.

Suppose $p$ has previously extracted $m$. Then $m$ is delivered by some process $q$ in round $r$. But $q$ has not previously extracted $m$. Hence $m$ is not delivered by any process in round $r$.

Claim 4: $p$ has not previously extracted $m$.

Suppose $p$ has previously extracted $m$. Then $m$ is delivered by some process $q$ in round $r$. But $q$ has not previously extracted $m$. Hence $m$ is not delivered by any process in round $r$.

Claim 5: $p$ has not previously extracted $m$.

Suppose $p$ has previously extracted $m$. Then $m$ is delivered by some process $q$ in round $r$. But $q$ has not previously extracted $m$. Hence $m$ is not delivered by any process in round $r$.

Lemma

If a correct process extracts $m$, then every correct process eventually extracts $m$. 

Agreement -2
Validity

- If the sender is correct, it executes broadcast(s,m,1) in round 1
- By CORRECTNESS, all correct processes execute accept(s,m,1) in round 1 and extract m
- In order to extract a different message m', a process must execute accept(s,m',i) in some round i\(\leq f+1\)
- By UNFORGEABILITY, and because s is correct, no correct process can extract m'
- All correct processes will deliver m

Implementing broadcast and accept

- A process that wants to broadcast m, does so through a series of witnesses
  - Sends m to all
  - Each correct process becomes a witness by relaying m to all
- If a process receives enough witness confirmations, it accepts m

Can we rely on witnesses?

- Only if not too many faulty processes!
- Otherwise, a set of faulty processes could fool a correct process by acting as witnesses of a message that was never broadcast
- How large can be f with respect to n?

Byzantine Generals

- One General, a set of Lieutenants
- General can order Attack or Retreat
- The General may be a traitor
- So may be some of the Lieutenants
- Devise a protocol by which:
  - If G is not a traitor, then all trustworthy L follow G’s battle plan
  - All trustworthy L agree on the battle plan
When can we solve it?

Suppose $n = 3$, and one traitor

A Lower Bound

Theorem
There is no algorithm that solves the terminating reliable broadcast problem for Byzantine failures if $n \leq 3$.

(Lamport, Shostak, and Pease. The Byzantine Generals Problem. ACM TOPLAS, 4 (3), 382-401, 1982)